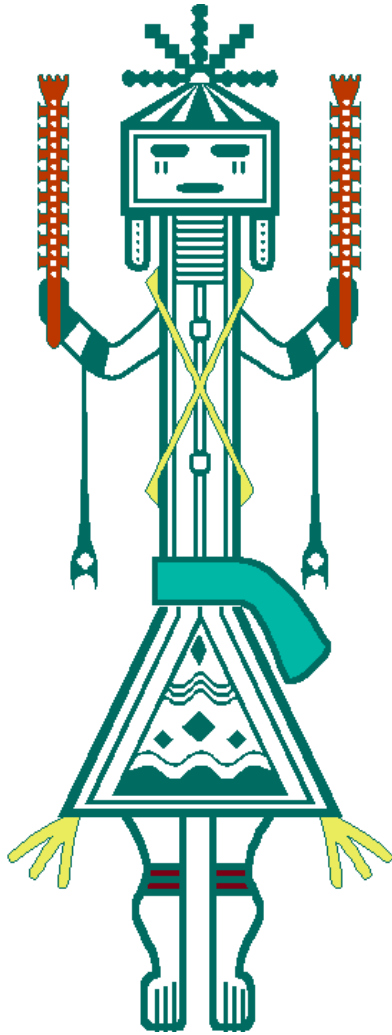


A Framework for Robust Engineering of Large-Scale Distributed Real-Time Systems

Dr. Connie U. Smith
L&S Computer Technology, Inc.
Performance Engineering Services
(505) 988-3811
www.spe-ed.com

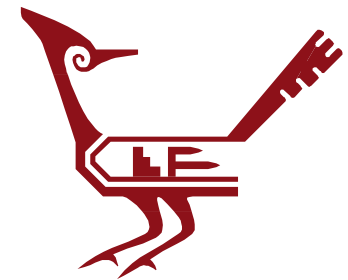
Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE APR 2010		2. REPORT TYPE		3. DATES COVERED 00-00-2010 to 00-00-2010	
4. TITLE AND SUBTITLE A Framework for Robust Engineering of Large-Scale Distributed Real-Time Systems				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) L&S Computer Technology, Inc, 7301 Burnet Road # 102, Austin, TX, 78757-2255				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES Presented at the 22nd Systems and Software Technology Conference (SSTC), 26-29 April 2010, Salt Lake City, UT. Sponsored in part by the USAF. U.S. Government or Federal Rights License					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 47	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

Overview

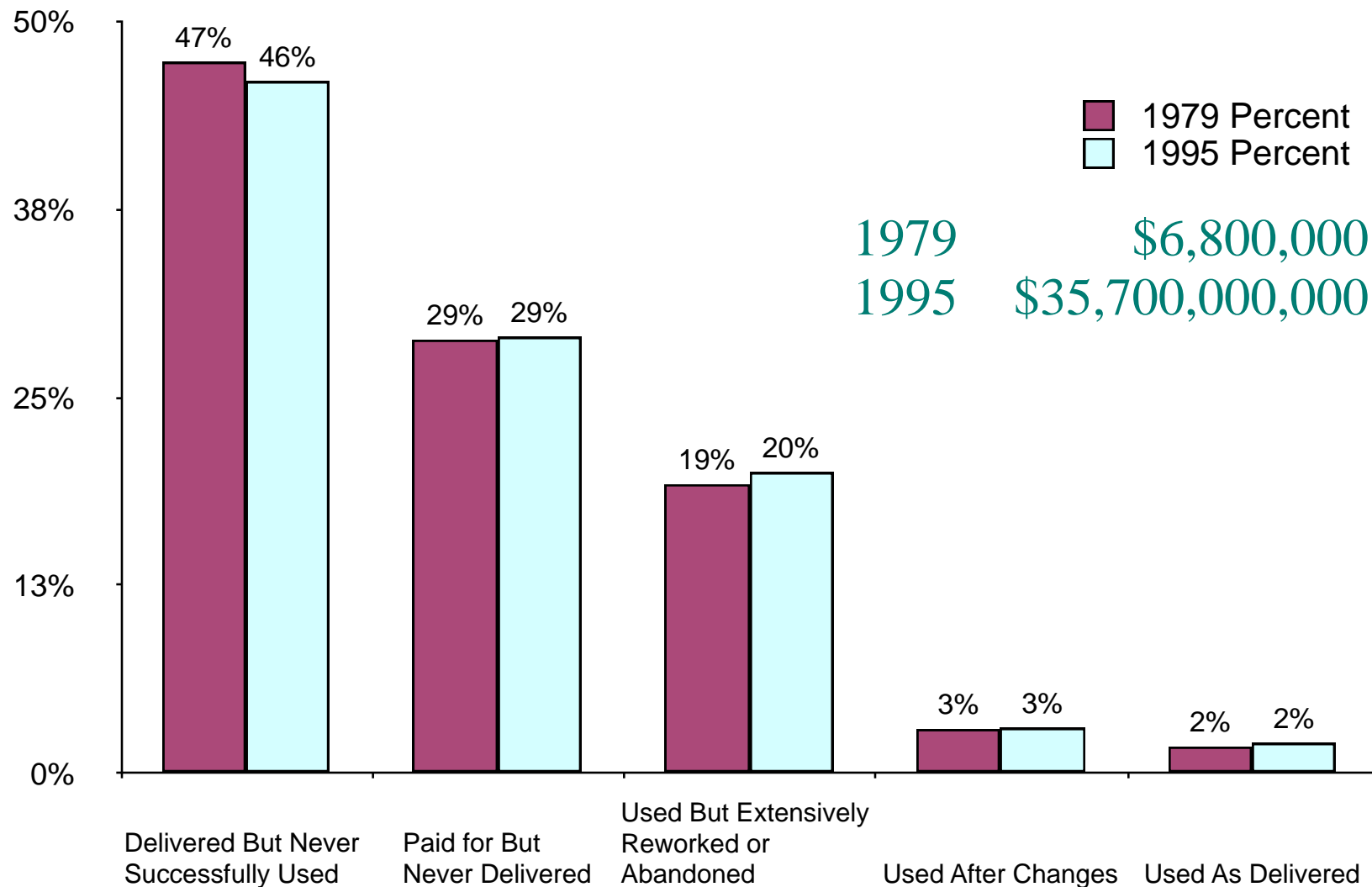


- ❖ Software Performance Engineering Overview
- ❖ Project Overview
- ❖ Phase 1 Accomplishments
- ❖ Status

Part 1: SPE Overview

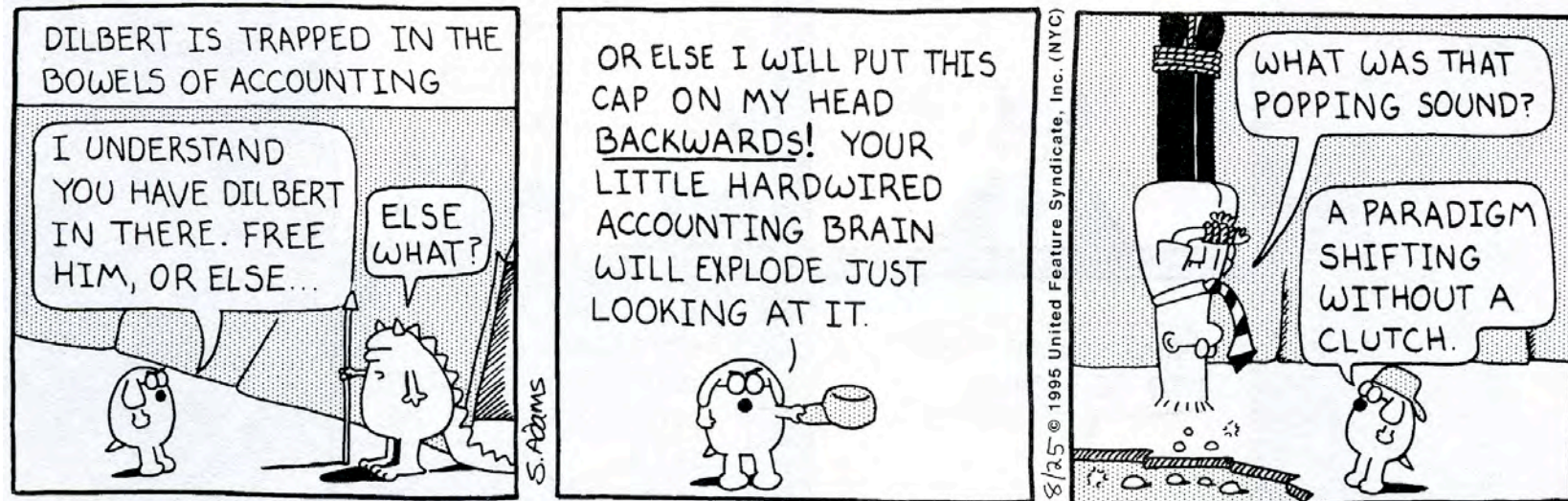


Federal Software Spending



L&S Computer Technology, Inc.©2010

A Paradigm Shift



"The significant problems we face cannot be solved at the same level of thinking we were at when we created them."

- Albert Einstein

The Dominant Paradigm

- ❖ Build and Test (Fix-It-Later)
 - ◆ “Let’s just build it and see what it will do.”
 - ◆ “You can’t do anything about performance until you have something to measure.”
- ❖ Improving the dominant paradigm
 - ◆ TQM or Six Sigma for testing
 - ◆ Do it faster
 - ◆ Strategic feasibility studies—“Best in class for testing/tuning.”
 - ◆ “Retreats” for testing team



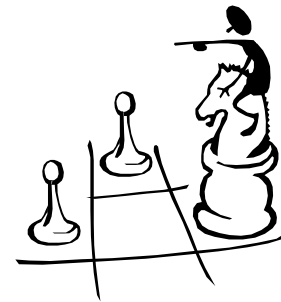
What's Wrong?

- ❖ The dominant paradigm is reactive
 - ◆ Finds problems, doesn't prevent them
 - ◆ Doesn't provide guidance for solving problems
 - ◆ Often finds problems when it is too late
 - ◆ Each problem is seen as unique



A "New" Paradigm

- ❖ A proactive approach to performance
- ❖ Early performance assessment and prediction
- ❖ Decision support for architects and designers
- ❖ Early identification and elimination of problems
- ❖ Guidelines and principles for
 - ◆ Preventing problems
 - ◆ Building-in performance



Why Worry About Performance?

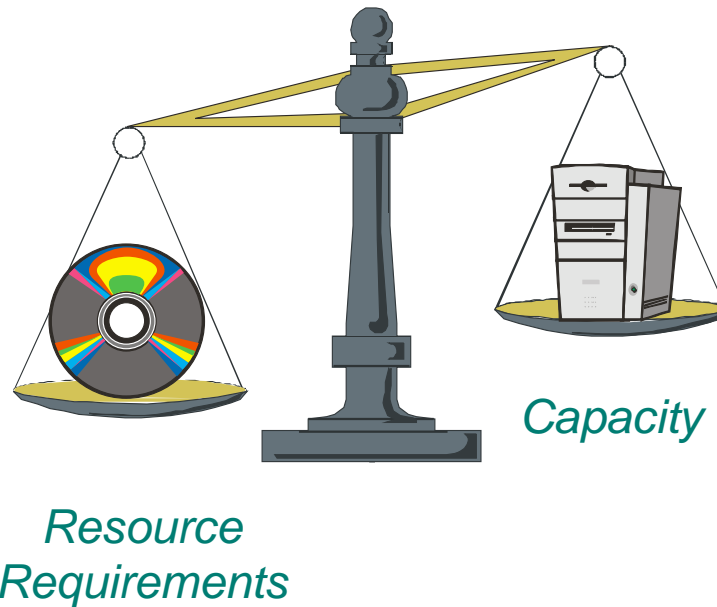
- ❖ Many systems cannot be used as initially implemented due to performance problems
- ❖ Problems are often due to fundamental architecture or design rather than inefficient code
 - ◆ Introduced early in development
 - ◆ Not discovered until late
- ❖ “Tuning” code after implementation
 - ◆ Disrupts schedules and creates negative user perceptions
 - ◆ Results in poorer overall performance (than building performance into architecture)
 - ◆ May not be possible to achieve requirements with tuning
 - ◆ Increases maintenance costs

Software Performance Engineering (SPE) Goal

- ❖ Early assessment of software decisions to determine their impact on quality attributes such as
 - ◆ performance
 - ◆ reliability
 - ◆ reusability
 - ◆ maintainability/modifiability
- ❖ Architecture has the most significant influence on quality attributes
- ❖ Architectural decisions are the most difficult to change



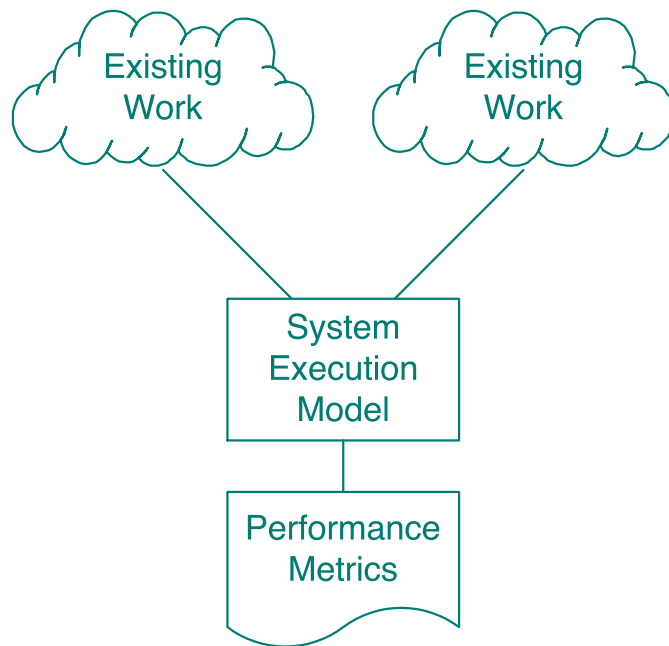
SPE Balance



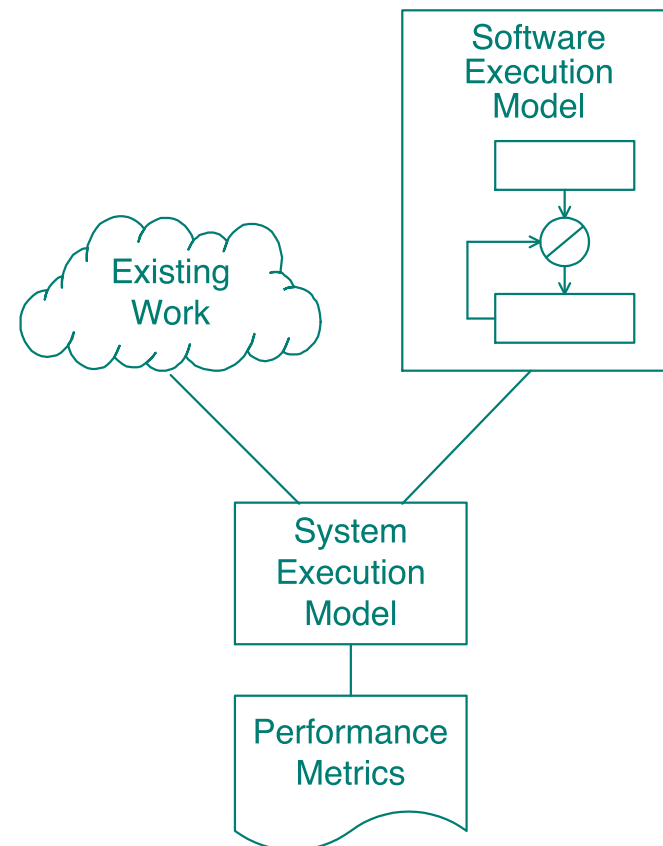
- ❖ Quantitative Assessment
- ❖ Begins early, frequency matches system criticality
- ❖ Often find architecture & design alternatives with lower resource requirements
- ❖ Select cost-effective performance solutions early

SPE Models

System Models



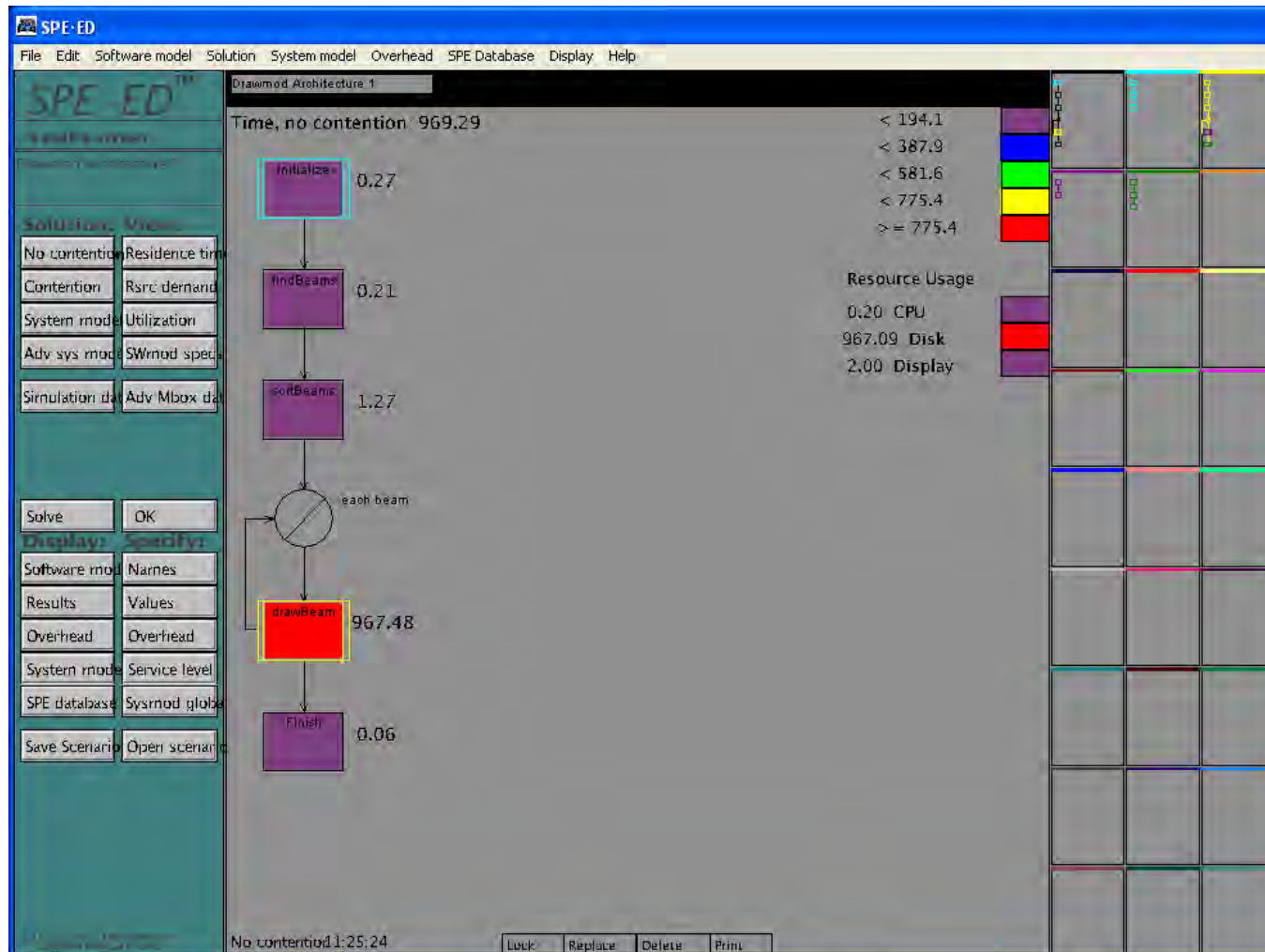
Software Prediction Models



SPE Model Requirements

- ❖ Low overhead
 - ◆ use the simplest possible model that identifies problems
- ❖ Accommodate:
 - ◆ incomplete definitions
 - ◆ imprecise performance specifications
 - ◆ changes and evolution
- ❖ Goals:
 - ◆ initially distinguish between "good" and "bad"
 - ◆ later, increase precision of predictions
 - ◆ provide decision support

SPE·ED



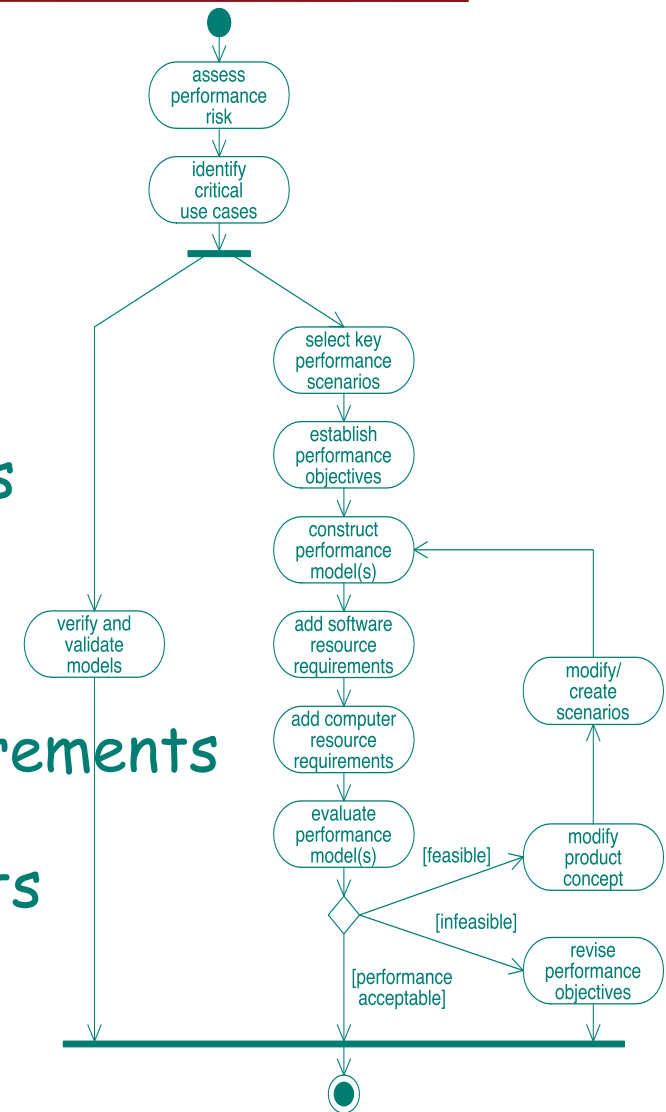
Established
technology

Customers

Source code

SPE Process Steps

1. Assess performance risk
2. Identify critical use cases
3. Select key performance scenarios
4. Establish performance requirements
5. Construct performance models
6. Determine software resource requirements
7. Add computer resource requirements
8. Evaluate the models
9. Verify and validate the models



Additional SPE Topics

- ❖ Performance Principles
- ❖ Performance Measurement
- ❖ Performance Patterns
- ❖ Architecture Assessment: PASASM
- ❖ Business Case for SPE
- ❖ SPE Best Practices
- ❖ SPE Metrics
- ❖ SPE Process

PERFORMANCE SOLUTIONS

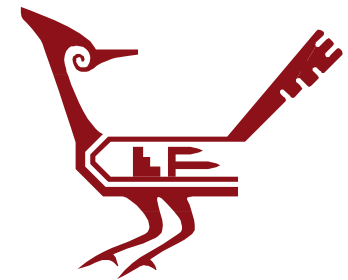
A PRACTICAL GUIDE TO CREATING
RESPONSIVE, SCALABLE SOFTWARE

CONNIE U. SMITH
LLOYD G. WILLIAMS

Forewords by Grady Booch and Paul Clements



Part 2: Model Interoperability Framework



Vision: Developers Do Robust Engineering

- ❖ Explore options using familiar tools & notations (UML)
- ❖ Select candidate designs for exploration
- ❖ Performance comparisons
 - ◆ Quantitative predictions from multiple tools
 - ◆ Performance metrics for software elements
 - ◆ Identify antipatterns
- ❖ Framework
 - ◆ Select metrics
 - ◆ Specify analysis conditions and select tools
 - ◆ Environment invokes analysis tool(s), collects output, prepares results in user-friendly format
- ❖ Bring in performance specialists for serious problems

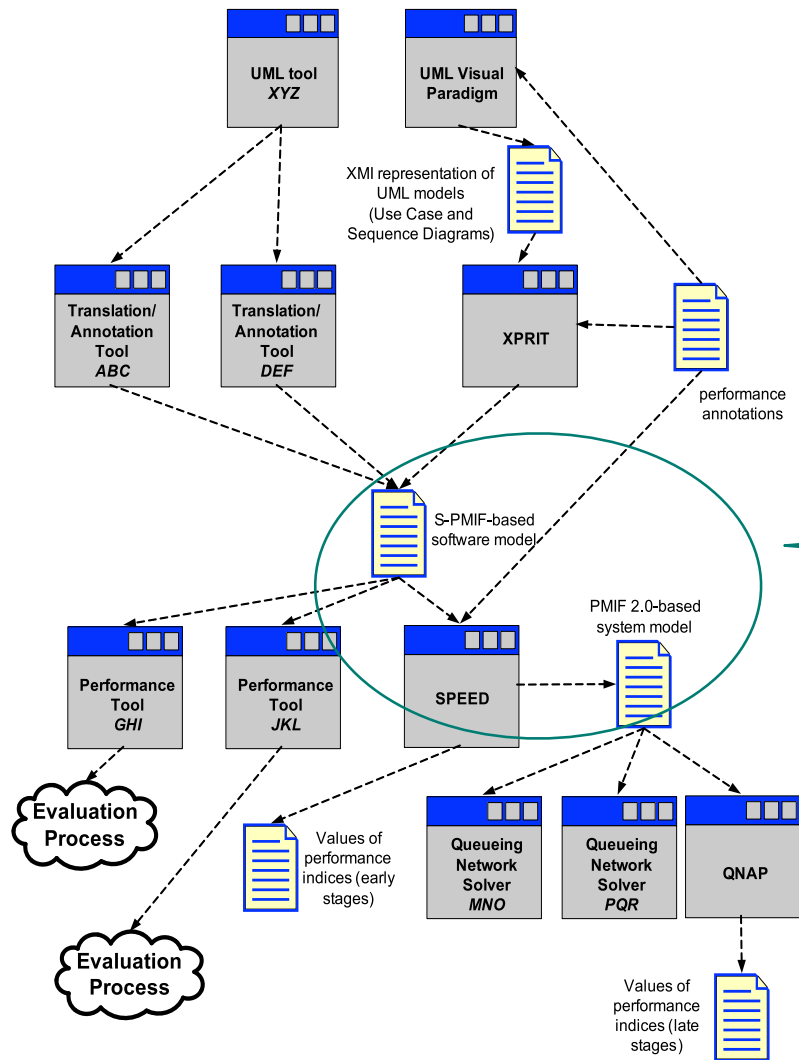
Motivation for Tool Interoperability

- ❖ Gap between software developers and performance specialists
- ❖ Economics/expertise required precludes building “tool for everything”
- ❖ Tools should specialize in what they do best and share knowledge with other tools

Our Research Strategy

- ❖ Bridge a variety of design and modeling tools
- ❖ Use software models as intermediate step to system performance models
- ❖ Re-use existing tools when appropriate
- ❖ De-skill the performance modeling & performance decision support
 - > empower developers who need performance info

UML Design Models -> Performance Models



Model
Interchange
Formats (MIFs)
streamline model
interoperability
process

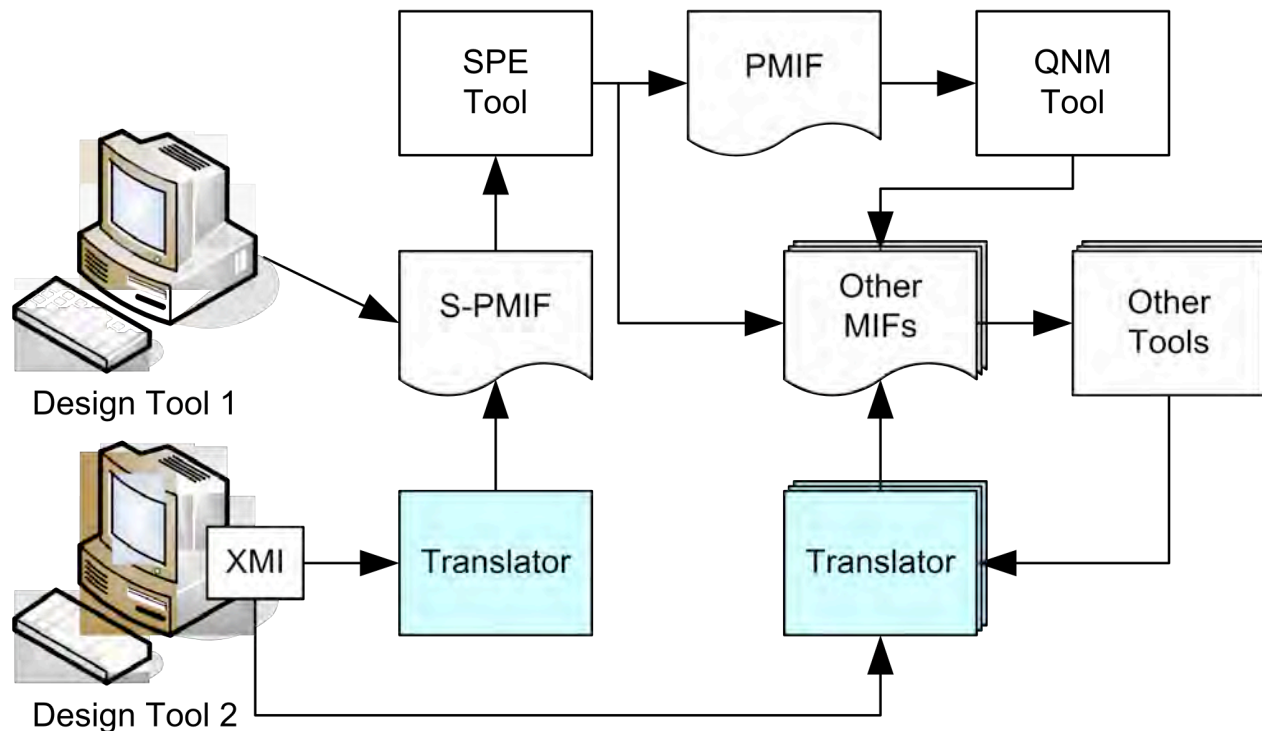
MIF Approach

- ❖ Common interface
 - ◆ No need for n^2 customized interfaces between tools
 - ◆ Import/export can be external to tools with file interfaces
- ❖ General approach to be used by a wide variety of tools
 - ◆ Meta-model of information requirements
 - ◆ Transfer format based on meta-model
- ❖ XML implementation
 - ◆ Meta-model -> schema, transfer format in XML
 - ◆ Relatively easy to create
 - ◆ XML is Verbose
 - but MIFs are a coarse grained interface
 - Exchange one file (not each individual element and attribute)

Our Research Results

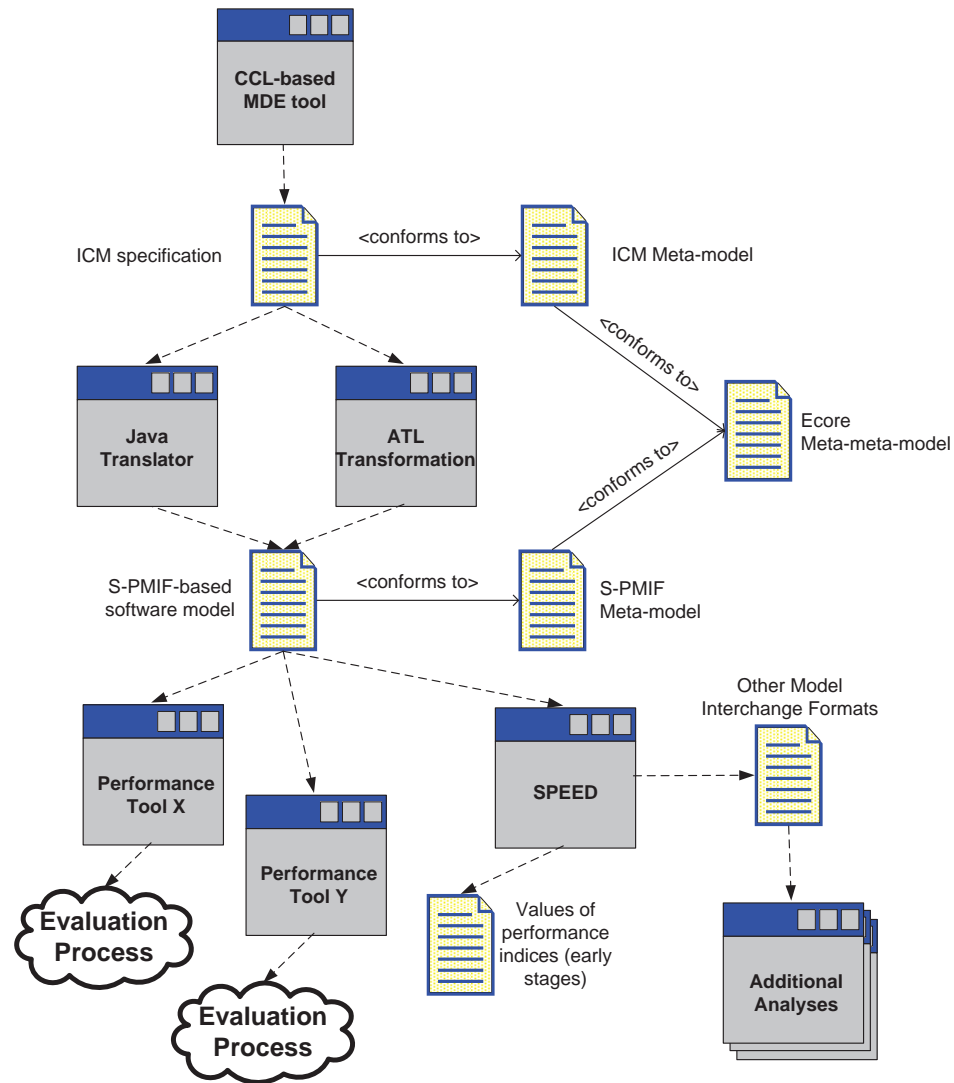
- ❖ Performance Model Interchange Format (PMIF)
 - ◆ Permit models defined in the standard format to be solved by all QNM modeling tools that support the standard
- ❖ Software Performance Model Interchange (S-PMIF)
 - ◆ Design tools to performance models
- ❖ Model solutions
 - ◆ Define a set of model runs independent of a given tool paradigm
 - Experiment Schema Extension (Ex-SE)
 - ◆ Define the output metrics desired from experiments
 - Output Schema Extension (Output-SE)
 - ◆ Define transformation from output to tables and charts
 - Results Schema Extension (Results-SE)

Our Current Approach - Several Distinct Steps

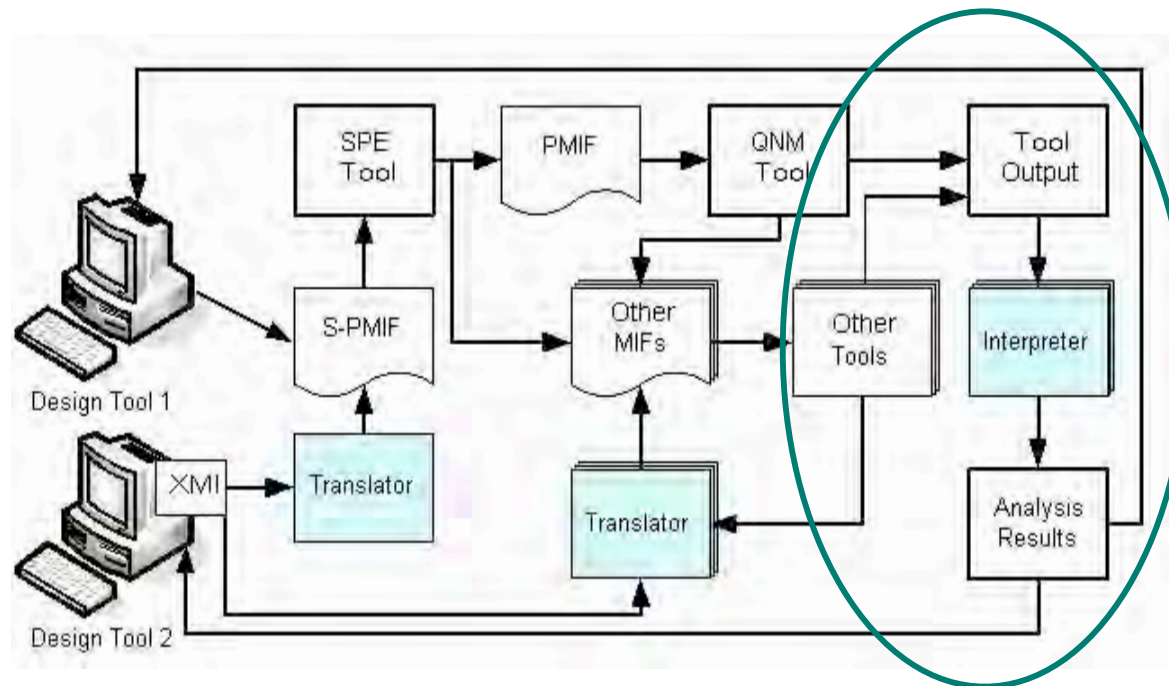


- ❖ A proof of concept has been implemented for each step
- ❖ Each step is a separate, independent program
- ❖ Expertise required limits usefulness for developers

Component Architecture -> Performance Models



Automated Approach for Developers



- ❖ Want to automate the end-to-end analysis steps:
 - ◆ Transformations, validation, experiment definition, and tool invocation,
 - ◆ Collect and present result data to developers for problem identification and diagnosis

Automate Performance Assessment of Software

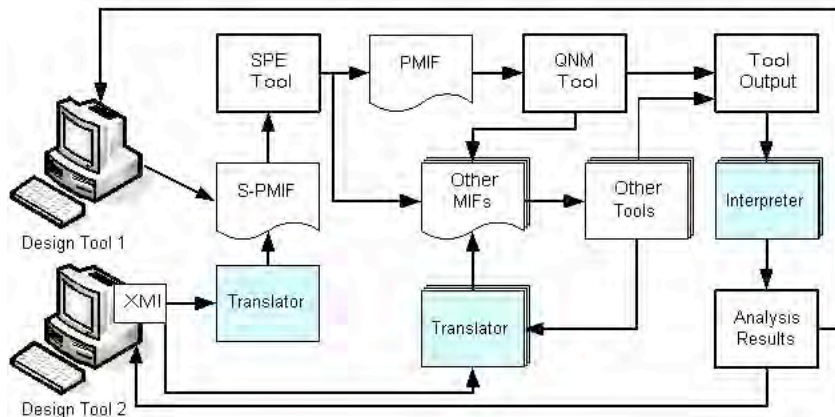
- ❖ Paradigm shift:
 - ◆ Enable developers to get quick performance analysis results without labor-intensive steps and
 - ◆ De-skill performance analysis steps to make SPE more available
- ❖ Streamline analysis
 - ◆ Keep models in sync as software evolves
 - ◆ Automated production of results
- ❖ Detect performance defects early
 - ◆ Easier and less costly to repair
- ❖ Increase likelihood of delivering useful software systems



Robust Engineering of Large Distributed RTES



Concept Diagram



Objective

- Robust Framework for automatic performance assessment of RTES
 - Translate designs to performance models
 - Define and execute experiments
 - Convert output metrics to meaningful results
 - Compare results from multiple tools
- Ability to extend Framework with new analysis capabilities for developers
 - Automated studies (scalability, sizing, sensitivity, etc.)
 - Identify problematic design features and performance antipatterns

Approach

- Build on our Model Interoperability Approach
 - Based on Performance Model Interchange Formats (PMIF and S-PMIF) and tool import and export interfaces
 - Complete enabling technologies for the Framework and support MARTE and MOF
- Define architecture for automatic integration of heterogeneous software design and performance analysis tools
 - Use Cases, User interface(s), automatic invocation of tools
- Develop Prototypes (Phase II)
 - Representative tools for end-to-end analysis from design to meaningful results
 - Mechanism for adding components to the Framework
- Demonstration – representative DoD RTES

Impact/Milestones

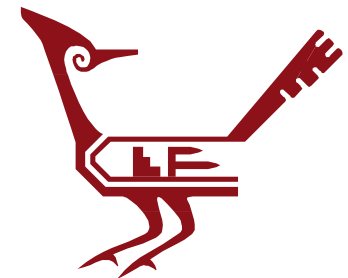
- FY09
 - Enabling technology complete
 - Architecture complete
- Improved analysis capabilities can cut up to 95% of time required for (manual) performance analysis of designs
- Automatically keep design and performance models in sync
 - Performance models keep pace with design changes
 - Eliminates manual comparison and re-creation of models
- Ease of use increases likelihood of conducting performance studies early in lifecycle
- Result: Better performing systems with optimally sized networks and platforms reduces hardware costs

Technology & Target Market: Analysts and Developers of Real-Time Embedded Systems (RTES)

Phase 1 Technical Objectives

1. **Define an architecture** that will support semi- to fully-automatic integration of heterogeneous software design and performance analysis tools.
2. **Align enabling technology** (S-PMIF and PMIF) with **MARTE** and **MOF**.
3. Investigate **improved analysis capabilities** for time-constrained large-scale systems deployed across a variety of communications and network topologies.
4. Develop a set of **Use Cases** to demonstrate the architecture's viability.
5. Define **sample user interface(s)** for selected Use Cases
6. Identify a representative, unclassified DoD **case study** for use in demonstrating the framework openness, scalability, and degree of automation during Phase II.
7. Identify an **initial set of design notations and tools** as well as analysis techniques and tools to be supported for the Phase II demonstration.
8. Develop a phased **implementation plan** for commercialization of the framework and plug-in tools, and incorporate it into **final report**.

Improved Analysis Capabilities: Model Output Metrics -> Useful results



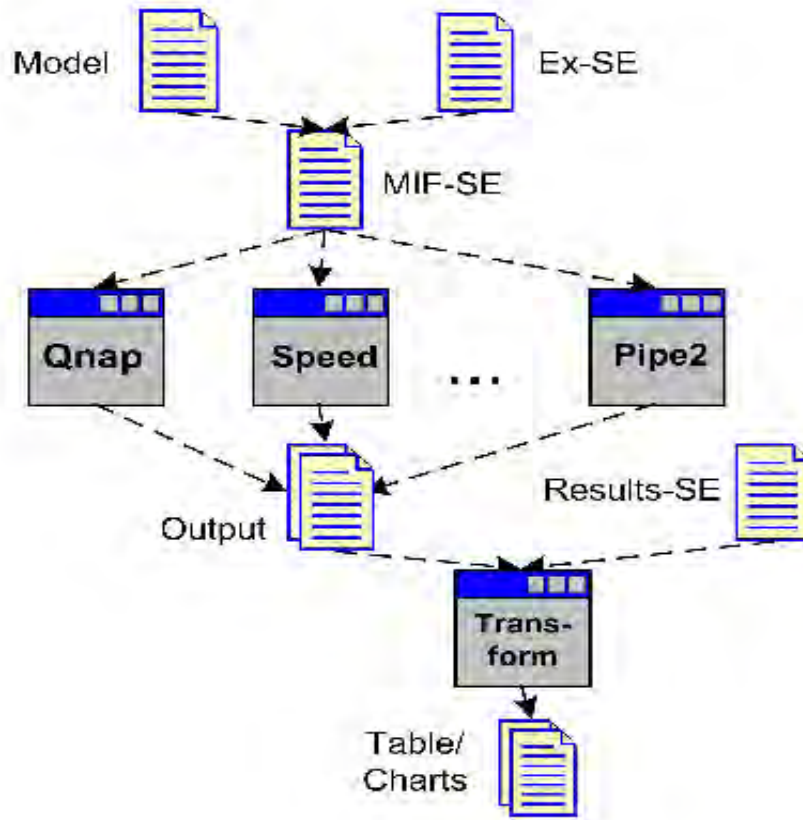
Assessment - Output -> Results

- ❖ Performance modeling tools produce numerical data
 - ◆ Output: Response times, utilizations, throughput, queue lengths, etc.
 - ◆ Users need a useful view of results
- ❖ Identified performance modeling Use Cases
- ❖ Surveyed output and results used in practice
 - ◆ Typical tables
 - ◆ Typical charts
 - ◆ Questions and answers (Q&A)

Requirements

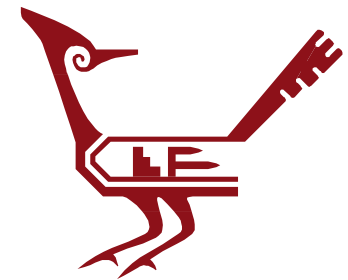
- ◆ Produce tables and charts for publication and presentation
- ◆ Streamline specification of common results
- ◆ Allow for creation and update
- ◆ Xls (Excel and OpenOffice) and LaTeX formats
- ◆ Allow for easy extension
- ◆ Visualization techniques are evolving
 - Include tool output reports with ToolCommand in the experiment specification
- ◆ Q&A deferred

Automated Experiments -> User Oriented Results

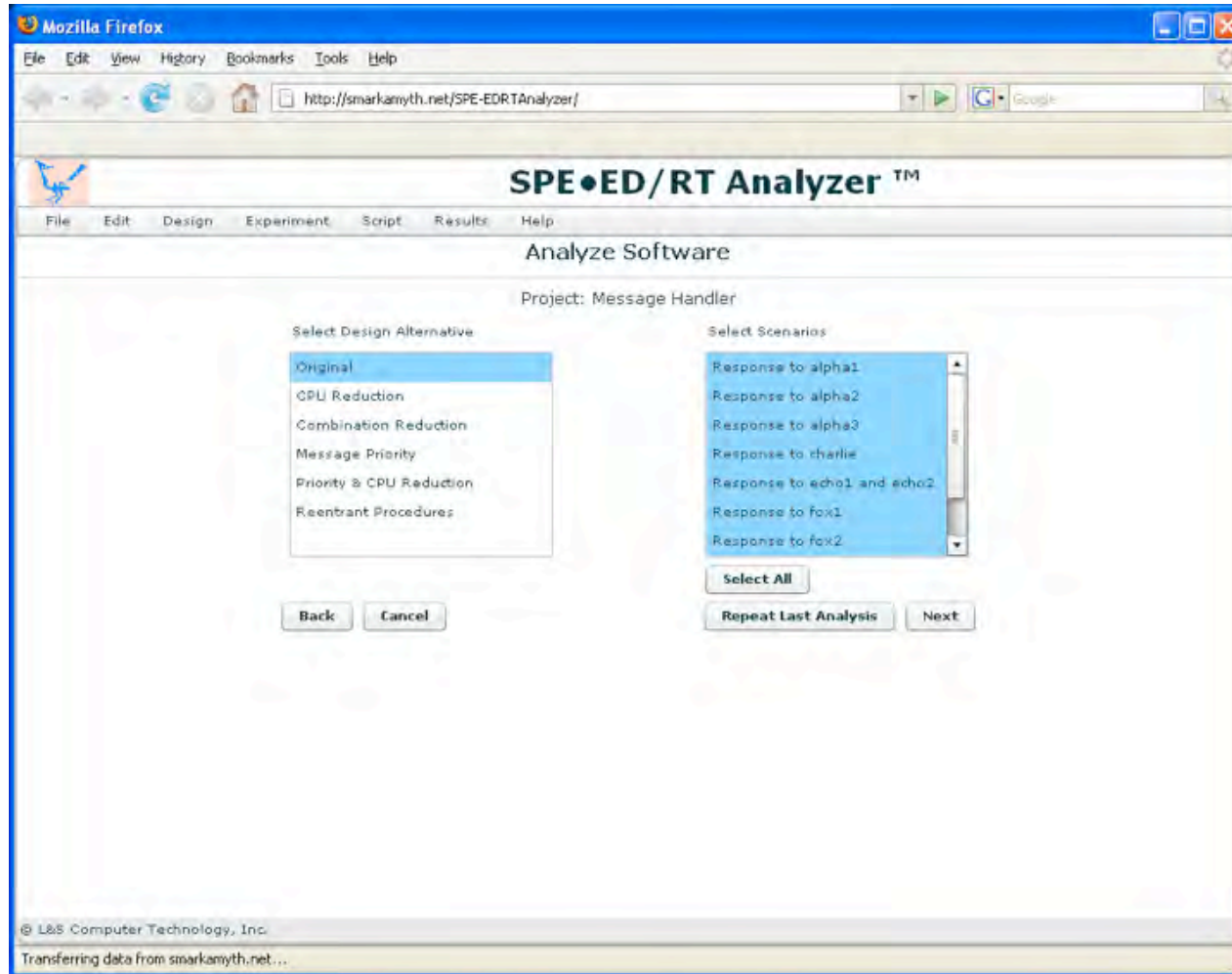


- ❖ Prototype transformation
- ❖ Output to xls
- ❖ Automatically re-produced complex tables
- ❖ Modeling paradigm-independent approach
- ❖ Customize to type of MIF

RT/Analyzer: Sample User Interface



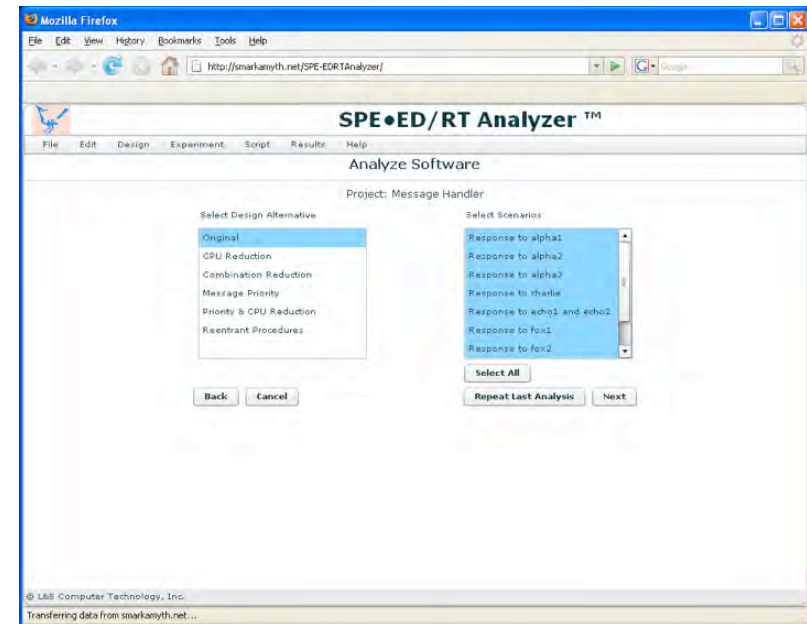
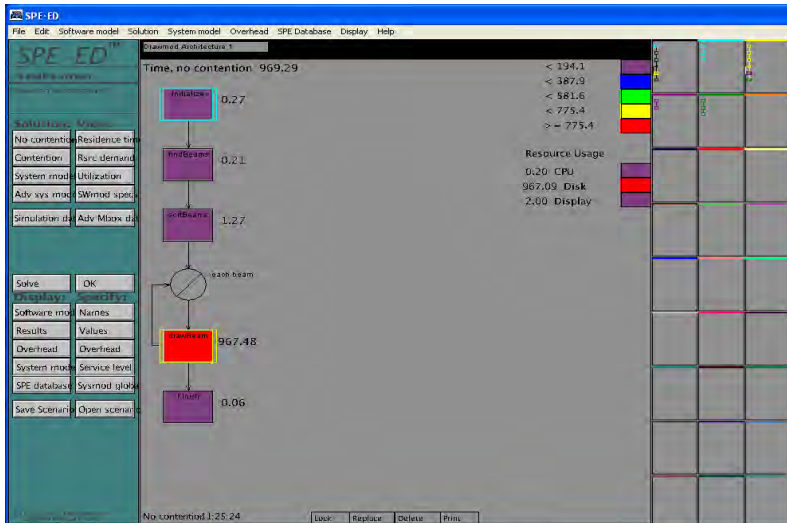
Clickable UI Demonstration



UI Demonstration

- ❖ Demonstrates ease of use for developers
- ❖ Selection of designs and experiments
- ❖ Meaningful results
- ❖ Flexbuilder foundation for Phase 2 implementation

SPE·ED -> RT/Analyzer



❖ SPE·ED

- ◆ Users are performance experts
- ◆ Primarily IT systems

❖ RT/Analyzer

- ◆ Target developers as users
- ◆ Focus on Real-Time Embedded System market sector

Phase I Successes: Enabling Technology

- ❖ Extensions for performance analysis of RTES
 - ◆ MARTE features to be supported
 - ◆ Model extensions for simulation solutions
- ❖ Improved analysis capabilities
 - ◆ Specification of automated model experiments
 - ◆ Transformation of model output into meaningful results
- ❖ Simplification of design translations
 - ◆ Meta-Object Facility (MOF) to enable model-to-model (M2M) transformations
 - ◆ Prototypes

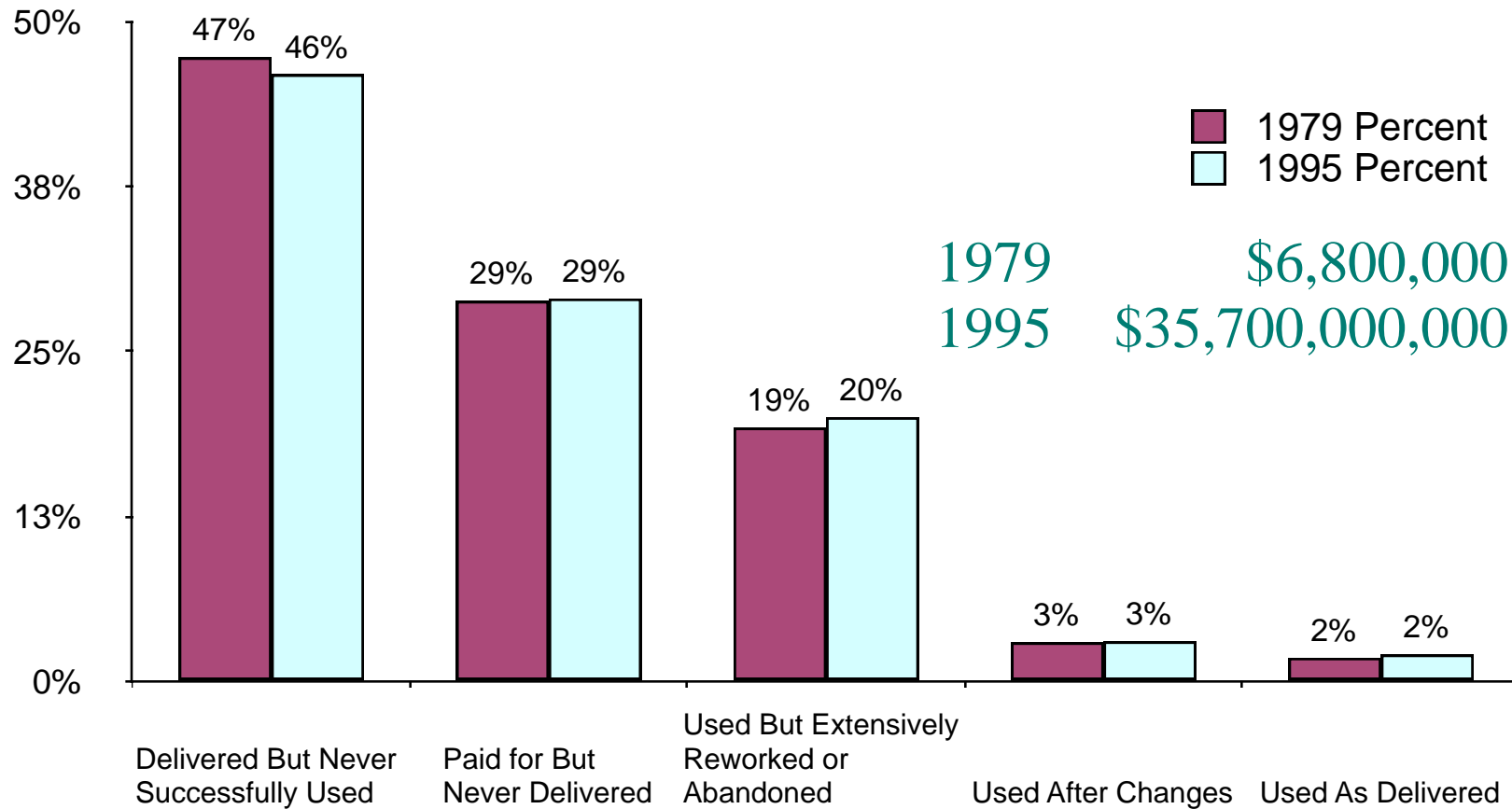
Phase I Successes: Tool Foundation

- ❖ Defined a model-interoperability architecture for RT/Analyzer
 - ◆ Use Cases and Scenarios
 - ◆ SOA Design Patterns incorporated into class diagram
- ❖ Proof of concept
 - ◆ Service prototypes
 - ◆ M2M translation for component architectures
 - ◆ Sample user interface
 - ◆ Case studies

Refereed Publications -> Technical Validity

1. "Automatic Generation of Performance Analysis Results: Requirements and Demonstration" *EPEW*, July 2009 (C.Smith, C. Lladó, UIB)
2. "Analysis of Real-Time Component Architectures: An Enhanced Model Interchange Approach," *Int. Journal Performance Evaluation* (C.Smith, G. Moreno, SEI)
3. "How to Automatically Transform Performance Model Output into Useful Results, " *CLEI*, Sept 2009 (C.Smith, C. Lladó, UIB)
4. "How to Automatically Execute Performance Models and Transform Output into Useful Results," *CMG*, Dec 2009 (C.Smith, C. Lladó, UIB)
5. "Software Performance Engineering: A Tutorial Introduction, *CMG*, Dec 2009 (C.Smith, L Williams)
6. "PMIF Extensions: Increasing the Scope of Supported Models," *Proc. WOSP*, San Jose, CA, Jan. 2010 (C.Smith, C. Lladó, R. Puigjaner)

P.S. Value of Problem Prevention



ROI if we can prevent performance problems

Lessons from history



Modernizing Telephone Switch Software

- ♦ Risk of new technology and/or inexperienced personnel
- ♦ Software Performance Antipattern
- ♦ Preventable with proper tools

RT/Analyzer Addresses Future Needs

❖ Cost

- ◆ Ability to predict performance of designs reduces cost of re-work due to late discovery of problems
- ◆ Up to 100 times more expensive to fix it later

❖ Quality

- ◆ Systems meet performance requirements

❖ Automated Analysis

- ◆ RT/Analyzer early detection of problems, performance ranking of solutions
- ◆ Less expertise and shorter time for analysis

❖ Productivity

- ◆ Quicker to build-in performance
- ◆ Resources can be devoted to development rather than re-work

Status

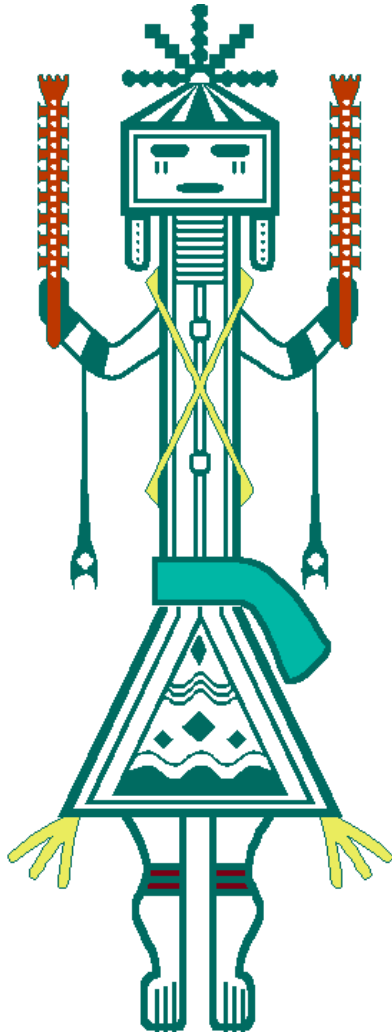
- ❖ RT/Analyzer architecture and enabling technology are positioned for future development
- ❖ Phase II funding not approved :-(
- ❖ Will continue development of RT/Analyzer but progress will be slower
- ❖ Still need comprehensive case study data

Conclusions



- ❖ Automated assessment of software and systems architecture is essential
 - ◆ We cannot continue to build RTES with today's methods
- ❖ RT/Analyzer is the right approach
 - ◆ Adaptable, extensible evolution
 - ◆ Model interoperability
- ❖ L&S Computer Technology is positioned to develop the tools
 - ◆ Performance expertise and vision
 - ◆ Software Performance Engineering market leaders

Summary



- ❖ Software Performance Engineering Overview
- ❖ Project Overview
- ❖ Phase 1 Accomplishments
- ❖ Status